ACEEE Int. J. on Information Technology, Vol. 3, No. 1, March 2013

Evolutionary Testing Approach for Solving Path-Oriented Multivariate Problems

P.Maragathavalli[1] and S.Kanmani[2]
[1]Pondicherry Engineering College /Department of Information Technology, Puducherry, India
E-mail:marapriya@pec.edu
[2]Pondicherry Engineering College /Department of Information Technology, Puducherry, India
E-mail:kanmani@pec.edu

***Abstract*— A multivariate approach involves varying number of objectives to be satisfied simultaneously in testing process. An evolutionary approach, genetic algorithm is taken for solving multivariate problems in software engineering. The Multivariate Optimization Problem (MOP) has a set of solutions, each of which satisfies the objectives at an acceptable level. Another evolutionary algorithm named SBGA (stage-based genetic algorithm) with two stages is attempted for solving problems with multiple objectives like cost minimization, time reduction and maximizing early fault deduction capabilities. In this paper, a multivariate genetic algorithm (MGA) in terms of stages for path-based programs is presented to get the benefits of both multi-criteria optimization and genetic algorithm. The multiple variables considered for test data generation are maximum path coverage with minimum execution time and test-suite minimization. The path coverage and the no. of test cases generated using SBGA are experimented with low, medium and high complexity object-oriented programs and compared with the existing GA approaches. The data-flow testing of OOPs in terms of path coverage are resulted with almost 88%. Thus, the efficiency of generated testcases has been improved in terms of path coverage with minimum execution time as well as with the minimal test suite size.**

***Index Terms*— multivariate problems, stage-based genetic algorithm, path coverage, execution time, test-suite reduction, evolutionary testing.**

I. Introduction

Evolutionary algorithms which are more advanced heuristic search techniques have been successfully applied in the area of software testing. For a large search space and getting optimal set of solutions GA is the best choice in software testing. Commonly, these techniques are referred as evolutionary testing. Evolutionary testing tries to improve the effectiveness and efficiency of the testing process by transforming testing objectives into search problems, and applying evolutionary similarity mutation is taken as new GA operators. Approaches, like ideal and preference are used in computation in order to solve them. In this testing of software can be done with a single objective or with multiple objectives [1][3]. Instead of fixing only one criteria or quality parameter for generating test cases, multiple objectives like minimizing the time & cost and no. of test cases simultaneously maximizing the coverage (i.e., the test requirements) would be considered. The techniques used in GA for multi-criteria optimization are of two types: one approach is sum mation of multiple fitness values named random weighted genetic algorithm (RWGA); another approach is considering multiple objectives at a time and generating Pareto optimal solution set named diversity based or non-dominated sorting genetic algorithm (NSGA) [4] [7] [8].

II. Existing Methods

Genetic algorithms are the most popular heuristic technique to solve Multi-Objective Optimization Problems [2]. A Multi-Objective Optimization Problem has a number of objective functions, which are to be minimized or maximized. MOOP can be expressed as, fitness function $f_m(x)$, where m = 1, 2, 3… M no. of objective functions & x = single candidate solution. The methods [1] [4] used for multi-objective optimization are ranking, diversity and combined approach, NRGA [5].

III. Stage-Based Genetic Algorithm (SBGA)

The design diagram of stage-based genetic algorithm by considering multiple objectives for stage-wise optimization is shown in Fig. 1. In this paper, the multi-objective optimization using SBGA is attempted by changing the GA operations with various GA parameter values. Instead of two-point crossover and mutation, uniform crossover and SBGA i.e. non-dominated sorting genetic algorithm [2] [6] which use two stages in generating test cases.The test case selection for next generation uses ranking of fitness values. By seeing the dominance between test cases, the test case is being selected for next iteration. In one-stage, sequencing of methods will take place and in another-stage the values will be passed to methods.

A. Input Representation

The input representation is in the form of control flow graphs (CFGs) and the output will be the generated test cases. CFG will give the dependencies between variables associated with specific nodes. *Control-flow graph of classes (CCFG)* For representing test objects in object-oriented programs by using CFGs in which a directed graph $G = (N, E)$ where nodes represent methods and edges represent procedure calls. The class control-flow graph (CCFG) is a class-call graph [2] [6] in which each node is replaced with the control-flow graph of the corresponding method and all methods of the class are connected together by edges.

Control-flow graph of classes (CCFG)

DOI: 01.IJIT.3.1. 1100

For representing test objects in object-oriented programs by using CFGs in which a directed graph $G = (N, E)$ where nodes represent methods and edges represent procedure calls. The class control-flow graph (CCFG) is a class-call graph [2] [6] in which each node is replaced with the control-flow graph of the corresponding method and all methods of the class are connected together by edges.

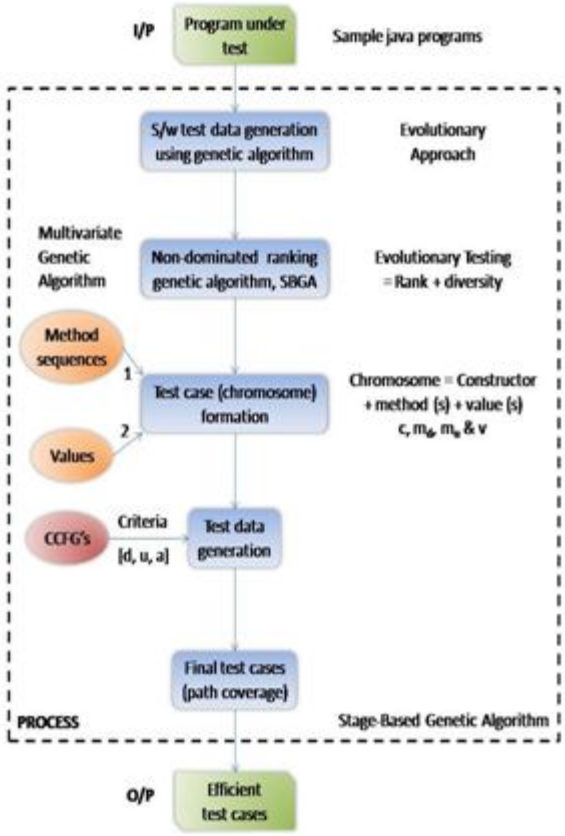


Fig. 1. Stage-Based Genetic Algorithm – An Evolutionary Approach

B. Test criteria and Chromosome Representation

The test criteria for generating test cases consists of a triple [d, u, a] where'd' def_node, 'u' use_node & 'a' associated variable. For example, consider the *stack* program with single class in Fig. 2; the criteria will be of the form [7, 11, top]. A test case consists of a sequence of methods invocation appended by their parameter values [1][2]. The 'chromosome' for a given criteria [d, u, a] contains the following:

- Invocation of a constructor 'c'
- Method m_d that causes the execution of 'd'
- Method m_u that causes the execution of 'u'
- Values to the parameters 'v'

C. Test Data Generation

The parameters and operators considered for test data generation in GA, their initial and range of values are tabulated in tableI and the results are shown in table II. The crossover used for getting the offsprings is uniform crossover which uses a fixed mixing ratio between two parents. For example, if the mixing ratio is 0.5, then half of the genes from both the parents go to offspring. The mutation is done using similarity mutation in which a similar gene is replaced with a different test case.

```
1   Class Stack
2   {
3   int top;
4   int a[] = new int[10];
      chromosome=constructor+method(s)+value(s)
      v_i = c m_d m_u & v
5   void create( )
6   {
7   top=0;
8   }
9   void push (int d)      Generation1:
10  {if(top==10)           test case1  1 3 &
11  return;                test case2  1 2 & 5
12  top = top + 1;         test case3 1 2 & 6
13  a[top] = d;Test criteria [d, u, a]  =  [20 18 top]
14  }
15  int pop( )Fitness- d – 16, 17,18, 20, u – 16, 17, 18
16  {   tc1 = 1/2(3/4) + 1/2(3/3) = 0.37 + 0.5 = 0.87
17  if (top==0)            tc2 = 0
18  return;                tc3 =0
19  int x = a[top];        Generation2:
20  top= top – 1;tc1 1 2 3&
    6=1/2(4/4)+1/2(3/3)=1.0
21  return x;  tc2 1 2 3 & 7 = 1/2(4/4)+1/2(3/3)  = 1.0
22  }
23  }                                  'I' = 2/3 =0.6
```

def-use associations

d	*u*	*a*
7	11	top
7	18	top
13	11	top
13	14	top
21	18	top
20	18	top

Fig. 2. Stack program as an example

TABLE I. GA OPERATORS

Name of the Parameter	Initial value	Range
Population size, M	40	40-200
Maximum generations per era, G	40	40-100
Number of eras, E	1	1-5
Crossover Probability, Cp	0.95	0.1-1.0
Mutation Probability, Mp	0.95	0.1-1.0

Fitness function $ft(v_i) = ½(cov\_d / tot\_d) + ½(cov\_u / tot\_u)$ * I = no. of chromosomes in next generation / no. of chromosomes in current generation

TABLE II. GA PARAMETERS

Name of the Operator	Type / value
Selection	Steady-state / fitness ≈ 1.0
Crossover	uniform / offsprings
Mutation	similarity / offsprings
Fitness function (*ft*)*	Float / (0.5 – 1.0)
Immigration rate (*I*)*	Float / (0.3 – 0.9)

DOI: 01.IJIT.3.1.1100

IV. Experiment

A. Test Data

Initially, simple java programs upto 200 lines of code are experimented with various coverage criterions. The results are satisfactory, and then five programs from Software Infrastructure Repository with equal complexity are taken for experiment; the lines of code are from the range (200-800). The no of classes are in the range of (1, 2, 4 … 24) and the branches of upto 60 are taken for experimentation. No. of generations is from 95 to 150 with the interval of 5; one era can have max. of 30 generations. The path coverage is predicted as a ratio between the conditions covered by the total no. of conditions in the tested program and presented in %.

B. Result Analysis

Programs like array-partition, ordset, replace, and schedule2 gives better coverage results in fast execution; because, the number of conditions to be checked for selecting test cases are less whereas in print-token2 and traffic collision avoidance system (TCAS) type of programs classes and conditions are more almost 60, so takes more time for execution. In *TCAS*, each and every vehicle should be monitored to order to avoid collision; so, takes more time to run the system with execution time of 4265ms in SBGA whereas GA takes 4912ms. The path coverage results are 97% in SBGA, GA covers only 87%. For *print-token2*, SBGA covers 96% with ET = 5140ms and GA 86% with 5990ms. The parameter Immigration rate, 'I' value should be minimum; since the 'I' value is taken from the difference between the current generation test cases (chromosomes) and the initial test cases, the value of new generation chromosomes should be minimum.

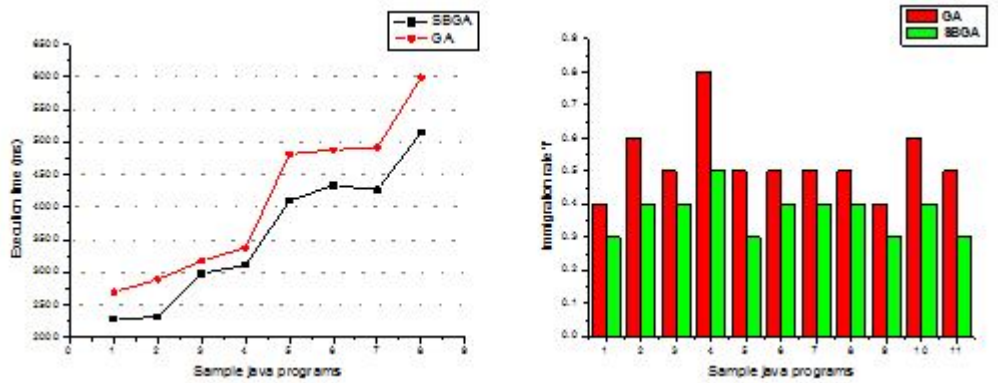
For example, when 10 test cases is reduced to 4, the value will be 0.4; reduced to 3, then 0.3; so, when I is less, small no of test cases are resulted; I is high, optimal set contains more no. of test cases; i.e. I = 4/10 = 0.4 ; I = 3/10 = 0.3; I = 5/10 = 0.5. The value 10 is only for an example; so, it need not be 10; 8/20 or some, 6/20 etc. On the whole, the no. of conditions satisfied by SBGA is more when compared to GA and is the results are tabulated in table III.

The execution time variation for both SBGA and GA is shown in graph 1; Test data generation time for each program varies from 2200 – 5990 ms. When no of classes and variables are more, the time is also be more; the conditions to be checked by test data increases, execution time will also be increased directly. Search time is a main criterion in GA for test data generation. So, that should be minimized. By using SBGA, the overall testing time is reduced by minimizing the total no of test cases generated. Reduction in test suite in terms of immigration rate 'I' is also shown in graph 1. Total no of chromosomes reduction is reflected in 'I'; initially, the rate

Table III. Results Obtained For Sample Java Programs

S. No	Sample programs	No. of classes	No. of test cases generated	Path coverage (cow/tot)		Execution time (ms)		Immigration rate (I)		Stopping criteria
				GA	SBGA	GA	SBGA	GA	SBGA	
1.	Array-partition	1	130	0.90	0.97	2700	2285	0.4	0.3	Tc = 4(reduced)
2.	Elevator	12	110	0.87	0.95	4815	4100	0.6	0.4	Upto 4 eras
3.	ordset	2	150	0.91	0.98	2890	2315	0.5	0.4	Till fitness=1.0
4.	deadlock	4	105	0.85	0.93	3278	2955	0.8	0.5	Upto 4 eras
5.	Alarm-clock	6	140	0.89	0.94	3565	3160	0.5	0.3	Till fitness=1.0
6.	Producer-consumer	8	100	0.90	0.96	3822	3305	0.5	0.4	Upto 4 eras
Java Programs from Software Infrastructure Repository (SIR)										
7.	TCAS	15	95	0.87	0.97	4912	4265	0.5	0.4	Tc = 4(reduced)
8.	Schedule2	13	110	0.90	0.98	4880	4335	0.5	0.4	Upto 4 eras
9.	Tot-info	4	140	0.89	0.96	3375	3110	0.4	0.3	Upto 4 eras
10.	Print-tokens2	24	120	0.86	0.96	5990	5140	0.6	0.4	Upto 6 secs
11.	Replace	2	115	0.89	0.98	3180	2985	0.5	0.3	Tc = 4(reduced)

DOI: 01.IJIT.3.1.1100



Graph. 1. Execution time in ms and Immigration rate 'I' results in SBGA and GA

is fixed as 0.3-0.9. In real scenario, SBGA ranges from 0.3-0.5 only; the comparison in immigration rate of SBGA with GA is also shown in graph1. So, the test data generated using stage-based genetic algorithm by simultaneously satisfying multiple objectives has been implemented and tested with different data sets.

Conclusion

Thus, the multivariate genetic algorithm with two stages SBGA is used for generation of object-oriented test cases. The fitness is purely depends on the path coverage of the test cases in the class. The results for sample java programs show that the efficiency and effectiveness of test cases generated by SBGA in terms of path coverage. In addition to path coverage, the time required for execution and the immigration rate are also satisfactory. This algorithm satisfies more than one variable in terms of objective simultaneously and tested with various sample program sets. The results can also be compared with other java packages for similar type of problems.

References

[1] Anon Sukstrienwong, "Solving multi-objective optimization under bounds by genetic algorithms," International Journal of Computers, Vol.5, Iss.1, pp. 18-25, 2011.

[2] Ahmed S. Ghiduk, "Automatic Generation of Object-Oriented Tests with a Multistage-Based Genetic Algorithm," Journal of computers, Vol. 5, No. 10, pp. 1560-1569, October 2010.

[3] Dhirendra Pal Singh, AshishKhare, "Different Aspects of Evolutionary Algorithms, Multi-Objective Optimization Algorithms and Application Domain," International Journal of Advanced Networking and Applications, Vol. 2, Iss.04, pp. 770-775, 2011.

[4] Abdullah Konak, David W. Coit, and Alice E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," Reliability Engineering and System Safety, Elsevier, pp. 992-1007, January 2006.

[5] Omar Al Jadaan, Lakishmi Rajamani, and C. R. Rao, "Non-dominated Ranking Genetic Algorithm for solving Multi-Objective Optimization Problems: NRGA," Journal of Theoretical and Applied Information Technology, pp. 60-67, 2008.

[6] Andreas S.Andreou, Kypros A. Economides and Anastasis A. Sofokleous, "An Automatic software test-data generation scheme based on data flow criteria and genetic algorithms," 7th International Conference on Computer and Information Technology, pp. 867-872, 2007.

[7] Kalyanmoy Deb, "Multi-Objective Optimization Using Evolutionary Algorithms: An Introduction," Technical Report 2011003, Indian Institute of Technology Kanpur, February 2011.

[8] Yuanyuan Zhang, "Multi-Objective Search-Based Requirements Selection and Optimization," Ph.D Thesis, University of London, pp. 1-276, February 2010.

DOI: 01.IJIT.3.1.1100